APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: DISCOURSE PARSING AND SUMMARIZATION

APPLICANT: DANIEL MARCU AND KEVIN KNIGHT

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL688320065US

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

May 11, 2001
Date of Deposit

Signature

Gildardo Vargas
Typed or Printed Name of Person Signing Certificate

TITLE

DISCOURSE PARSING AND SUMMARIZATION

Related Application

This application claims the benefit of, and incorporates herein, U.S. Provisional Patent Application No. 60/203,643, filed May 11, 2000.

Origin of Invention

The research and development described in this application were supported by the NSA under grant number MDA904-97-0262 and by DARPA / ITO under grant number MDA904-99-C-2535. The US government may have certain rights in the claimed inventions.

Field of the Invention

The present application relates to computational linguistics and more particularly to techniques for parsing a text to determine its underlying rhetorical, or discourse, structure, and to techniques for summarizing, or compressing, text.

Background and Summary

Computational linguistics is the study of the applications of computers in processing and analyzing language, as in automatic machine translation ("MT") and text analysis. In

conjunction with MT research and related areas in computational linguistics, researchers have developed and frequently use various types of tree structures to graphically represent the structure of a text segment (e.g., clause, sentence, paragraph or entire treatise). Two basic tree types include (1) the syntactic tree, which can be used to graphically represent the syntactic relations among components of a text segment, and (2) the rhetorical tree (equivalently, the rhetorical structure tree (RST) or the discourse tree), which can be used to graph the rhetorical relationships among components of a text segment. Rhetorical structure trees are discussed in detail in William C. Mann and Sandra A. Thompson, "Rhetorical structure theory: Toward a functional theory of text organization," Text, 8(3):243-281 (1988) (hereinafter, "Mann and Thompson (1988)"). Discourse tree structures find application in many areas including machine translation, summarization, information retrieval, automatic test scoring and the like.

The example in Fig. 1 shows the types of structures in a discourse tree 100 for a text fragment. The leaves 102 of the tree correspond to elementary discourse units ("edus") and the internal nodes correspond to contiguous text spans. Each node in a discourse tree is characterized by a "status" (i.e., either "nucleus" or "satellite") and a "rhetorical relation," which is a relation that holds between two non-overlapping text spans. In

Fig. 1, nuclei 104 are represented by straight lines while satellites 106 are represented by arcs.

The distinction between nuclei and satellites comes from empirical observations that a nucleus expresses information that is more essential than a satellite to the writer's intention, and that the nucleus of a rhetorical relation is comprehensible independent of the satellite but not vice versa. When spans are equally important, the relation is said to be "multinuclear."

Rhetorical relations reflect semantic, intentional and/or textual relations that hold between text spans. Examples of rhetorical relations include the following types indicated in capitals: one text span may ELABORATE on another text span; the information in two text spans may be in CONTRAST; and the information in one text span may provide JUSTIFICATION for the information presented in another text span. Other types of rhetorical relations include EVIDENCE, BACKGROUND, JOINT, and CAUSE. In Fig. 1, the internal nodes of discourse tree 100 are labeled with their respective rhetorical relation names 108.

In conventional practice, discourse trees either have been generated by hand by trained personnel or have been pieced together in a semi-automated manner using manually generated instructions for a computer program. Development of the discourse parsing systems and techniques described below was based in part on the recognition that manually generating

discourse trees in either of these fashions is time-consuming, expensive and prone to inconsistencies and error. Accordingly, a computer-implemented discourse parsing system and automated discourse parsing techniques were developed for automatically generating a discourse tree for any previously unseen text segment based on a set of automatically learned decision rules.

Implementations of the disclosed discourse parsing system and techniques may include various combinations of the following features.

In one aspect, a discourse structure for an input text segment (e.g., a clause, a sentence, a paragraph or a treatise) is determined by generating a set of one or more discourse parsing decision rules based on a training set, and determining a discourse structure for the input text segment by applying the generated set of discourse parsing decision rules to the input text segment.

The training set may include a plurality of annotated text segments (e.g., built manually by human annotators) and a plurality of elementary discourse units (*edus*). Each annotated text segment may be associated with a set of *edus* that collectively represent the annotated text segment.

Generating the set of discourse parsing decision rules may include iteratively performing one or more operations (e.g., a shift operation and one or more different types of reduce

operations) on a set of *edus* to incrementally build the annotated text segment associated with the set of *edus*. The different types of reduce operations may include one or more of the following six operations: *reduce-ns, reduce-sn, reduce-nn, reduce-below-ns, reduce-below-sn, reduce-below-nn.* The six reduce operations and the shift operation may be sufficient to derive the discourse tree of any input text segment.

Determining a discourse structure may include incrementally building a discourse tree for the input text segment, for example, by selectively combining elementary discourse trees (*edts*) into larger discourse tree units. Moreover, incrementally building a discourse tree for the input text segment may include performing operations on a stack and an input list of *edts*, one *edt* for each *edu* in a set of *edus* corresponding to the input text segment.

Prior to determining the discourse structure for the input text segment, the input text segment may be segmented into *edus*, which are inserted into the input list. Segmenting the input text segment into *edus* may be performed by applying a set of automatically learned discourse segmenting decision rules to the input text segment. Generating the set of discourse segmenting decision rules may be accomplished by analyzing a training set.

Determining the discourse structure for the input text segment may further include segmenting the input text segment

into elementary discourse units (*edus*); incrementally building a discourse tree for the input text segment by performing operations on the *edus* to selectively combine the *edus* into larger discourse tree units; and repeating the incremental building of the discourse tree until all of the *edus* have been combined.

In another aspect, text parsing may include generating a set of one or more discourse segmenting decision rules based on a training set, and determining boundaries in an input text segment by applying the generated set of discourse segmenting decision rules to the input text segment. Determining boundaries may include examining each lexeme in the input text segment in order, and, for example, assigning, for each lexeme, one of the following designations: sentence-break, *edu*-break, start-parenthetical, end-parenthetical, and none. More generally, determining boundaries in the input text segment may include recognizing sentence boundaries, *edu* boundaries, parenthetical starts, and parenthetical ends. Examining each lexeme in the input text segment may include associating features with the lexeme based on surrounding context.

In another aspect, generating discourse trees may include segmenting an input text segment into *edus*, and incrementally building a discourse tree for the input text segment by performing operations on the *edus* to selectively combine the *edus*

into larger discourse tree units. The incremental building of the discourse tree may be repeated until all of the *edus* have been combined into a single discourse tree. Moreover, the incremental building of the discourse tree is based on predetermined decision rules, such as automatically learned decision rules generated by analyzing a training set of annotated discourse trees.

In another aspect, a discourse parsing system may include a plurality of automatically learned decision rules; an input list comprising a plurality of *edts*, each *edt* corresponding to an *edu* of an input text segment; a stack for holding discourse tree segments while a discourse tree for the input text segment is being built; and a plurality of operators for incrementally building the discourse tree for the input text segment by selectively combining the EDTs into a discourse tree segment according to the plurality of decision rules and moving the discourse tree segment onto the stack. The system may further include a discourse segmenter for partitioning the input text segment into *edus* and inserting the *edus* into the input list.

One or more of the following advantages may be provided by discourse parsing systems and techniques as described herein. The systems and techniques described here result in a discourse parsing system that uses a set of learned decision rules to automatically determine the underlying discourse structure of any

unrestricted text. As a result, the discourse parsing system can be used, among other ways, for constructing discourse trees whose leaves are sentences (or units that can be identified at high levels of performance). Moreover, the time, expense, and inconsistencies associated with manually built discourse tree derivation rules are reduced dramatically.

The ability to automatically derive discourse trees is useful not only in its standalone form (e.g., as a tool for linguistic researchers) but also as a component of a larger system, such as a discourse-based machine translation system. Accordingly, the systems and techniques described herein represent an enabling technology for many different applications including text, paragraph or sentence summarization, machine translation, informational retrieval, test scoring and related applications.

The rhetorical parsing algorithm described herein implements robust lexical, syntactic and semantic knowledge sources. Moreover, the six reduce operations used by the parsing algorithm, along with the shift operation, are mathematically sufficient to derive the discourse structure of any input text.

Text summarization (also referred to as text compression) is the process of a taking a longer unit of text (e.g., a long sentence, a paragraph, or an entire treatise) and converting it into a shorter unit of text (e.g., a short sentence or an

abstract) referred to as a summary. Automated summarization - that is, using a computer or other automated process to produce a summary - has many applications, for example, in information retrieval, abstracting, automatic test scoring, headline generation, television captioning, and audio scanning services for the blind. Fig. 10 shows a block diagram of an automated summarization process. As shown therein, an input text 1000 is provided to a summarizer 1002, which generates a summary 1004 of the input text 1000.

Ideally, whether produced manually or automatically, a summary will capture the most salient aspects of the longer text and present them in a coherent fashion. For example, when humans produce summaries of documents, they do not simply extract sentences, clause or keywords, and then concatenate them to form a summary. Rather, humans attempt to summarize by rewriting the longer text, for example, by constructing new sentences that are grammatical, that cohere with one another, and that capture the most salient items of information in the original document.

Conventional attempts at automated summarization, in contrast, typically have focused on identifying relevant items of information in the text being summarized, extracting text segments (e.g., sentences, clauses or keywords) corresponding to those identified items, and then concatenating together the

extracted segments. Moreover, these conventional approaches typically rely on manually generated sets of summarization rules.

Development of the summarizing systems and techniques described below was based in part on the recognition (1) that identification, extraction and concatenation of relevant text segments typically will not generate a coherent and/or grammatical summary and/or (2) that manually generated summarization rules are prone to error and inconsistencies, are time-consuming and expensive to generate, and generally result in non-ideal summaries. Accordingly, as described in detail below, automated summarization systems and techniques were developed that can generate a coherent summary of an input text by generating new, grammatical sentences that capture the salient aspects of the input text.

Implementations of the disclosed summarization systems and techniques may include various combinations of the following features.

In one aspect, a tree structure (e.g., a discourse tree or a syntactic tree) is summarized by generating a set of one or more summarization decision rules (e.g., automatically learned decision rules) based on a training set, and compressing the tree structure by applying the generated set of summarization decision rules to the tree structure. The tree structure to be compressed may be generated by parsing an input text segment such as a

clause, a sentence, a paragraph, or a treatise. The compressed tree structure may be converted into a summarized text segment that is grammatical and coherent. Moreover, the summarized text segment may include sentences not present in a text segment from which the pre-compressed tree structure was generated.

Applying the generated set of summarization decision rules comprises performing a sequence of modification operations on the tree structure, for example, one or more of a shift operation, a reduce operation, and a drop operation. The reduce operation may combine a plurality of trees into a larger tree, and the drop operation may delete constituents from the tree structure.

The training set used to generate the decision rules may include pre-generated long/short tree pairs. Generating the set of summarization decision rules comprises iteratively performing one or more tree modification operations on a long tree until the paired short tree is realized. A plurality of long/short tree pairs may be processed to generate a plurality of learning cases. In that case, generating the set of decision rules may include applying a learning algorithm to the plurality of learning cases. Moreover, one or more features may be associated with each of the learning cases to reflect context.

In another aspect, a computer-implemented summarization method may include generating a parse tree (e.g., a discourse tree or a syntactic tree) for an input text segment, and

iteratively reducing the generated parse tree by selectively eliminating portions of the parse tree. Iterative reduction of the parse tree may be performed based on a plurality of learned decision rules, and may include performing tree modification operations on the parse tree. The tree modification operations may include one or more of the following: a shift operation, a reduce operation (which, for example, combines a plurality of trees into a larger tree), and a drop operation (which, for example, deletes constituents from the tree structure).

In another aspect, summarization is accomplished by parsing an input text segment to generate a parse tree (e.g., a discourse tree or a syntactic tree) for the input segment, generating a plurality of potential solutions, applying a statistical model to determine a probability of correctness for each of potential solution, and extracting one or more high-probability solutions based on the solutions' respective determined probability of correctness. Applying a statistical model may include using a stochastic channel model algorithm that, for example, performs minimal operations on a small tree to create a larger tree. Moreover, using a stochastic channel model algorithm may include probabilistically choosing an expansion template. Generating a plurality of potential solutions may include identifying a forest of potential compressions for the parse tree.

The generated parse tree may have one or more nodes, each

node having N children (wherein N is an integer). In that case, identifying a forest of potential compressions may include generating $2^N - 1$ new nodes, one node for each non-empty subset of the children, and packing the newly generated nodes into a whole. Alternatively, or in addition, identifying a forest of potential compressions may include assigning an expansion-template probability to each node in the forest.

Extracting one or more high-probability solutions may include selecting one or more trees based on a combination of each tree's word-bigram and expansion-template score. For example, a list of trees may be selected, one for each possible compression length. The potentials solutions may be normalized for compression length. For example, for each potential solution, a log-probability of correctness for the solution may be divided by a length of compression for the solution.

One or more of the following advantages may be provided by summarization systems and techniques as described herein.

The systems and techniques described here result in a summarization system that can take virtually any longer text segment (sentence, phrase, paragraph or treatise) and compress it into a shorter version that is both grammatical and coherent. In contrast to the conventional "extract and concatenate" summarization techniques, the disclosed summarizer generates new

grammatical sentences that more closely resemble summarizations prepared by trained human editors.

Moreover, the disclosed summarizer generates summaries automatically, e.g., in a computer-implemented manner. Accordingly, the inconsistencies, errors, time and/or expense typically incurred with conventional approaches that require manual intervention are reduced dramatically.

The two different embodiments of the summarizer (channel-based and decision-based) both generate coherent, grammatical results but also potentially provide different advantages. On the one hand, the channel-based summarizer provides multiple different solutions at varying levels of compression. These multiple solutions may be desirable if, for example, the output of the summarizer was being provided to a user (e.g., human or computer process) that could make use of multiple outputs. On the other hand, the decision-based summarizer is deterministic and thus provides a single solution and does so very quickly. Accordingly, depending on the objectives of the user, the decision-based summarizer may be advantageous both for its speed and for its deterministic approach.

Moreover, the channel-based summarizer may be advantageous depending on a user's objectives because its performance can be adjusted, or fine-tuned, to a particular application by replacing or adjusting its statistical model. Similarly, performance of

the decision-based summarizer can be fine-tuned to a particular application by varying the training corpus used to learn decision rules. For example, a decision-based summarizer could be tailored to summarize text or trees in a specific discipline by selecting a training corpus specific to that discipline.

The details of one or more embodiments are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the description and drawings, and from the claims.

Drawing Descriptions

The above and other aspects will now be described in detail with reference to the accompanying drawings, wherein:

Fig. 1 shows and example of a discourse tree.

Fig. 2 is a flowchart of generating a discourse tree for an input text.

Fig. 3 is a block diagram of a discourse tree generating system.

Fig. 4 shows an example of shift-reduce operations performed in discourse parsing a text.

Fig. 5 shows the operational semantics of six reduce operations.

Fig. 6 is a flowchart of generating decision rules for a discourse segmenter.

Fig. 6A shows examples of automatically derived segmenting rules.

Fig. 7 is a graph of a learning curve for a discourse segmenter.

Fig. 8 is a flowchart of generating decision rules for a discourse segmenter.

Fig. 8A shows examples of automatically derived shift-reduce rules.

Fig. 8B shows a result of applying Rule 1 in Fig. 8A on the *edts* that correspond to the units in text example (5.1).

Fig. 8C shows a result of applying Rule 2 in Fig.8A on the *edts* that correspond to the units in text example (5.2).

Fig. 8D shows an example of a CONTRAST relation that holds between two paragraphs.

Fig. 8E shows a result of applying Rule 4 in Fig.8A on the on the trees that subsume the two paragraphs in Fig. 8D.

Fig. 9 is a graph of a learning curve for a shift-reduce action identifier.

Fig. 10 is a block diagram of an automated summarization system.

Fig. 11 shows examples of parse (or syntactic) trees.

Fig. 12 shows examples of text from a training corpus.

Fig. 13 is a graph of adjusted log-probabilities for top scoring compressions at various compression lengths.

Fig. 14 shows an example of incremental tree compression.

Fig. 15 shows examples of text compression.

Fig. 16 shows examples of summarizations of varying compression lengths.

Fig. 17 is a flowchart of a channel-based summarization process.

Fig. 18 is a flowchart of a process for training a channel-based summarizer.

Fig. 18A shows examples of rules that were learned automatically by the C4.5 program

Fig. 19 is a flowchart of a decision-based summarization process.

Fig. 20 is a flowchart of a process for training a decision-based summarizer.

Detailed Description

DISCOURSE PARSING

As described herein, a decision-based rhetorical parsing system (equivalently, a discourse parsing system) automatically derives the discourse structure of unrestricted texts and incrementally builds corresponding discourse trees based on a set of learned decision rules. The discourse parsing system uses a shift-reduce rhetorical parsing algorithm that learns to construct rhetorical structures of texts from a corpus of

discourse-parse action sequences. The rhetorical parsing algorithm implements robust lexical, syntactic and semantic knowledge sources.

In one embodiment, the resulting output of the discourse parsing system is a rhetorical tree. This functionality is useful both in its standalone form (e.g., as a tool for linguistic researchers) and as a component of a larger system, such as in a discourse-based machine translation system, as described in Daniel Marcu et al., "The Automatic Translation of Discourse Structures," Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics, pp. 9-17, Seattle, Washington, (April 29 -- May 3, 2000), and Daniel Marcu, "The Theory and Practice of Discourse Parsing and Summarization," The MIT Press (2000), both of which are incorporated herein.

Fig. 2 shows a flowchart of a discourse parsing process 200 that generates a discourse tree from an input text. Upon receiving the input text in step 202, the process 200 breaks the text into elementary discourse units, or "edus." *Edus* are defined functionally as clauses or clause-like units that are unequivocally the nucleus or satellite of a rhetorical relation that holds between two adjacent spans of text. Further details of *edus* are discussed below.

Next, in step 206, the *edus* are put into an input list. In step 208, the process 200 uses the input list, a stack, and a set of learned decision rules to perform the shift-reduce rhetorical parsing algorithm, which eventually yields the discourse structure of the text given as input. In step 210, performing the algorithm results in the generation of a discourse tree that corresponds to the input text.

Fig. 3 shows a block diagram of a discourse tree generating system 300 that takes in input text 301 and produces discourse tree 305. The system 300 as shown includes two sub-systems: (1) a discourse segmenter 302 that identifies the *edus* in a text, and (2) a discourse parser 304 (equivalently, a shift-reduce action identifier), which determines how the *edus* should be assembled into rhetorical structure trees.

The discourse segmenter 302, which serves as a front-end to the discourse parser 304, partitions the input text into edus. The discourse segmenter processes an input text one lexeme (word or punctuation mark) at a time and recognizes sentence and *edu* boundaries and beginnings and ends of parenthetical units.

The discourse parser 304 takes in the *edus* from the segmenter 302 and applies the shift-reduce algorithm to incrementally build the discourse tree 305. As indicated in Fig. 3, in this embodiment, each of the discourse segmenter 302 and the discourse parser 304 performs its operations based on a set

of decision rules that were learned from analyzing a training set, as discussed in detail below. An alternative embodiment is possible, however, in which substantially the same results could be achieved using probabilistic rules.

Further details of the discourse parser and the parsing process that it performs are provided with reference to Figs. 4 and 5. Details on generating discourse segmenter decision rules and discourse parser decision rules appear below with reference to Figs. 6-9. What follows is a description of the training corpus that was used in generating decision rules for the discourse segmenter and for the discourse parser.

The Training Corpus

The training corpus (equivalently, the training set) used was a body of manually built (i.e., by humans) rhetorical structure trees. This corpus, which included 90 texts that were manually annotated with discourse trees, was used to generate learning cases of how texts should be partitioned into *edus* and how discourse units and segments should be assembled into discourse trees.

A corpus of 90 rhetorical structure trees were used, which were built manually using rhetorical relations that were defined informally in the style of Mann et al., "Rhetorical structure theory: Toward a functional theory of text organization, *Text*,

8(3):243-281 (1988): 30 trees were built for short personal news stories from the MUC7 co-reference corpus (Hirschman et al., *MUC-7 Coreference Task Definition*, 1997); 30 trees for scientific texts from the Brown corpus; and 30 trees for editorials from the Wall Street Journal (WSJ). The average number of words for each text was 405 in the MUC corpus, 2029 in the Brown corpus and 878 in the WSJ corpus. Each MUC text was tagged by three annotators; each Brown and WSJ text was tagged by two annotators.

The rhetorical structure assigned to each text is a (possibly non-binary) tree whose leaves correspond to *elementary discourse units (edu)*s, and whose internal nodes correspond to contiguous text spans. Each internal node is characterized by a *rhetorical relation*, such as ELABORATION and CONTRAST. Each relation holds between two non-overlapping text spans called NUCLEUS and SATELLITE. (There are a few exceptions to this rule: some relations, such as SEQUENCE and CONTRAST, are multinuclear.) As noted above, the distinction between nuclei and satellites comes from the empirical observation that the nucleus expresses what is more essential to the writer's purpose than the satellite. Each node in the tree is also characterized by a promotion set that denotes the units that are important in the corresponding subtree. The promotion sets of leaf nodes are the leaves themselves. The promotion sets of internal nodes are

given by the union of the promotion sets of the immediate nuclei nodes.

As noted above, *edus* are defined functionally as clauses or clause-like units that are unequivocally the NUCLEUS or SATELLITE of a rhetorical relation that holds between two adjacent spans of text. For example, "because of the low atmospheric pressure" in the text (1), below, is not a fully fleshed clause. However, since it is the SATELLITE of an EXPLANATION relation, it is treated as elementary.

(1) [Only the midday sun at tropical latitudes is warm enough] [to thaw ice on occasion,] [but any liquid water formed in this way would evaporate almost instantly] [because of the low atmospheric pressure.]

Some *edus* may contain *parenthetical units*, i.e., embedded units whose deletion does not affect the understanding of the *edu* to which they belong. For example, the unit shown in italics in text (2), below, is parenthetic.

(2) This book, *which I have received from John*, is the best book that I have read in a while.

The annotation process involved assigning *edu* and parenthetical unit boundaries, assembling *edus* and spans into discourse trees, and labeling the relations between *edus* and spans with rhetorical relation names from a taxonomy of 71 relations. No explicit distinction was made between intentional,

informational, and textual relations. In addition, two constituency relations were marked that were ubiquitous in the corpus and that often subsumed complex rhetorical constituents. These relations were ATTRIBUTION, which was used to label the relation between a reporting and a reported clause, and APPOSITION. The rhetorical tagging tool used - namely, the RST Annotation Tool downloadable from, and described at:

http://www.isi.edu/~marcu/software.html

maintains logs of all tree-construction operations. As a result, in addition to the rhetorical structure of 90 texts, a corpus of logs was created that reflects the way that human judges determine *edu* and parenthetical unit boundaries. The following two publications - Daniel Marcu, Estibaliz Amorrortu, and Magdalena Romera, "Experiments in Constructing a Corpus of Discourse Trees," The ACL'99 Workshop on Standards and Tools for Discourse Tagging, Maryland, June 1999; and Daniel Marcu, Magdalena Romera, and Estibaliz Amorrortu, "Experiments in Constructing a Corpus of Discourse Trees: Problems, Annotation Choices, Issues," The Workshop on Levels of Representation in Discourse, pages 71-78, Edinburgh, Scotland, July 1999 - both of which are incorporated by reference, discuss in detail the annotation tool and protocol and assess the inter-judge agreement and the reliability of the annotation.

The Discourse Parsing Model

The discourse parsing process is modeled as a sequence of shift-reduce operations. The input to the parser is an empty stack and an input list that contains a sequence of elementary discourse trees ("edts"), one edt for each *edu* produced by the discourse segmenter. The status and rhetorical relation associated with each edt is "UNDEFINED", and the promotion set is given by the corresponding edu. At each step, the parser applies a "Shift" or a "Reduce" operation. Shift operations transfer the first edt of the input list to the top of the stack. Reduce operations pop the two discourse trees located on the top of the stack; combine them into a new tree updating the statuses, rhetorical relation names, and promotion sets associated with the trees involved in the operation; and push the new tree on the top of the stack.

Assume, for example, that the discourse segmenter partitions a text given as input as shown in text (3) below (only the *edus* numbered from 12 to 19 are shown):

(3) … [Close parallels between tests and practice tests are common, [12]] [some educators and researchers say.[13]] [Test preparation booklets, software and worksheets are a booming publishing subindustry.[14]] [But some practice products are so similar to the tests themselves that critics say they represent a form of school-sponsored cheating.[15]]

["If they took these preparation booklets into my classroom,[16]] [I'd have a hard time justifying to my students and parents that it wasn't cheating,"[17]] [says John Kaminsky,[18]] [a Traverse City, Mich., teacher who has studied test coaching.[19].] …

Fig. 4 shows the actions taken by a shift-reduce discourse parser starting with step *i*. At step *i*, the stack contains 4 partial discourse trees, which span units [1,11], [12,15], [16,17], and [18], and the input list contains the edts that correspond to units whose numbers are higher than or equal to 19. At step *i* the parser decides, based on its predetermined decision rules, to perform a Shift operation. As a result, the edt corresponding to unit 19 becomes the top of the stack. At step *i+1*, the parser performs a "Reduce-Apposition-NS" operation, that combines edts 18 and 19 into a discourse tree whose nucleus is unit 18 and whose satellite is unit 19. The rhetorical relation that holds between units 18 and 19 is APPOSITION. At step *i+2*, the trees that span over units [16,17] and [18,19] are combined into a larger tree, using a "Reduce-Attribution-NS" operation. As a result, the status of the tree [16,17] becomes "nucleus" and the status of the tree [18,19] becomes "satellite." The rhetorical relation between the two trees is SMALL ATTRIBUTION. At step *i+3*, the trees at the top of the stack are combined using a

"Reduce-Elaboration-NS" operation. The effect of the operation is shown at the bottom of Fig. 4.

In order to enable a shift-reduce discourse parser to be able to derive any discourse tree, it is sufficient to implement one Shift operation and six types of Reduce operations, whose operational semantics are shown in Fig. 5. In other words, the shift operation and the six reduce operations shown in Fig. 5 are mathematically sufficient to derive the discourse tree of any unrestricted input text.

For each possible pair of nuclearity assignments "nucleus-satellite" (ns), "satellite-nucleus" (sn), and "nucleus-nucleus" (nn) there are two possible ways to attach the tree located at position *top* in the stack to the tree located at position *top-1*. To create a binary tree whose immediate children are the trees at *top* and *top-1*, an operation of type "reduce-ns", "reduce-sn", or "reduce-nn" is used. To attach the tree at position *top* as an extra-child of the tree at *top-1*, thus creating or modifying a non-binary tree, an operation of type "reduce-below-ns", "reduce-below-sn", or "reduce-below-nn" is used. Fig. 5 illustrates how the statuses and promotion sets associated with the trees involved in the reduce operations are affected in each case.

Because the labeled data in the training corpus used was relatively sparse, the relations that shared some rhetorical meaning were grouped into clusters of rhetorical similarity. For

example, the cluster named "contrast" contained the contrast-like rhetorical relations of ANTITHESIS, CONTRAST, and CONCESSION. The cluster named "evaluation-interpretation" contained the rhetorical relations EVALUATION and INTERPRETATION. And the cluster named "other" contained rhetorical relations such as question-answer, proportion, restatement, and comparison, which were used very seldom in the corpus. The grouping process yielded 17 clusters, each characterized by a generalized rhetorical relation name. These names are as follows: APPOSITION-PARENTHETICAL, ATTRIBUTION, CONTRAST, BACKGROUND-CIRCUMSTANCE, CAUSE-REASON-EXPLANATION, CONDITION, ELABORATION, EVALUATION-INTERPRETATION, EVIDENCE, EXAMPLE, MANNER-MEANS, ALTERNATIVE, PURPOSE, TEMPORAL, LIST, TEXTUAL, and OTHER.

If a sufficiently large number of texts were labeled manually, however, the clustering described above would be unnecessary.

In developing the discourse parser, one design parameter was to automatically derive rhetorical structures trees that were labeled with relation names that corresponded to the 17 clusters of rhetorical similarity. Since there are 6 types of reduce operations and since each discourse tree uses relation names that correspond to the 17 clusters of rhetorical similarity, it follows that the discourse parser needs to learn what operation

to choose from a set of 6 X 17 + 1 = 103 operations (the 1 corresponds to the SHIFT operation).

The Discourse Segmenter

Fig. 6 is a flowchart of a generalized process 600 for generating decision rules for the discourse segmenter. The first step in the process was to build, or otherwise obtain, the training corpus. As discussed above, this corpus was built manually using an annotation tool. In general, human annotators looked at text segments and for each lexeme (word or punctuation mark) determined whether an *edu* boundary existed at the lexeme under consideration and either marked it with a segment break or not, depending on whether an *edu* boundary existed.

Next, in step 604, for each lexeme, a set of one or more features was associated to each of the *edu* boundary decisions, based on the context in which these decisions were made. The result of such association is a set of learning cases - essentially, discrete instances that capture the *edu*-boundary decision-making process for a particular lexeme in a particular context. More specifically, the leaves of the discourse trees that were built manually were used in order to derive the learning cases. To each lexeme in a text, one learning case was associated using the features described below. The classes to be learned, which are associated with each lexeme, are "sentence-

break", "edu-break", "start-paren", and "end-paren", and "none". Further details of the features used in step 604 for learning follow.

To partition a text into *edus* and to detect parenthetical unit boundaries, features were relied on that model both the local and global contexts. The local context consists of a window of size 5 (1 + 2 +2) that enumerates the Part-Of-Speech (POS) tags of the lexeme under scrutiny and the two lexemes found immediately before (2) and after it (2). The POS tags are determined automatically, using the "Brill Tagger," as described in Eric Brill, "Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging," Computational Linguistics, 21(4):543–565, which is incorporated by reference. Because discourse markers, such as "because" and "and", typically play a major role in rhetorical parsing, also considered was a list of features that specify whether a lexeme found within the local contextual window is a potential discourse marker; hence, for each lexeme under scrutiny, it is specified whether it is a special orthographical marker, such as comma, dash, and parenthesis, or whether it is a potential discourse marker, such as "accordingly," "afterwards," and "and." The local context also contains features that estimate whether the lexemes within the window are potential

abbreviations. In this regard, a hard-coded list of 250 potential abbreviations can be used.

The global context reflects features that pertain to the boundary identification process. These features specify whether there are any commas, closed parentheses, and dashes before the estimated end of the sentence, whether there are any verbs in the unit under consideration, and whether any discourse marker that introduces expectations was used in the sentence under consideration. These markers include phrases such as *Although* and *With*.

The decision-based segmenter uses a total of twenty-five features, some of which can take as many as 400 values. When we represent these features in a binary format, we obtain learning examples with 2417 binary features/example.

In step 606, a learning algorithm such as the C4.5 algorithm as described in J. Ross Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufmann Publishers (1993), to learn a set of decision rules from the learning cases. The result is set of discourse segmenter decision rules 608 that collectively define whether a previously unseen lexeme, given its particular context, represents an *edu* boundary within its particular context in the text segment under consideration.

Figure 6A shows some of the rules that were learned by the C4.5 program using a binary representation of the features and

learning cases extracted from the MUC corpus. Rule 1 specifies that if the POS tag of the lexeme that immediately precedes the lexeme under scrutiny is a closed parenthesis and the previous marker recognized during the processing or the current sentence was an open parenthesis, then the action to be taken is to insert an end of parenthetic unit. Rule 1 can correctly identify the end of the parenthetic unit at the location marked with the symbol ↑ in sentence (4.1) below.

(4.1) Surface temperatures typically average about -60 degrees Celsius (-76 degrees Fahrenheit) ↑ at the equator.

Rule 2 can correctly identify the beginning of the parenthetic unit 44 years old in sentence 4.2 because the unit is preceded by a comma and starts with a numeral (CD) followed by a plural noun (NNS).

(4.2) Ms. Washington, ↑ 44 years old, would be the first woman and the first black to head the five-member commission that oversees the securities markets.

Rule 3 identifies the end of a sentence after the occurrence of a DOT (period, questions mark, or exclamation mark) that is not preceded or followed by another DOT and that is not followed by a DOUBLEQUOTE. This rule will correctly identify the sentence end after the period in example 4.3, but will not insert a sentence end after the period in example 4.4. However, another

rule that is derived automatically will insert a sentence break after the double quote that follows the ↑ mark in example 4.4.

(4.3) The meeting went far beyond Mr. Clinton's normal weekly gathering of business leaders. ↑ Economic advisor Gene Sperling described it as "a true full-court press" to pass the deficit-reduction bill, the final version of which is now being hammered out by House and Senate Negotiators.

(4.4) The executives "are here, just as I am, not because anyone agrees with every last line and jot and title of this economic program," Mr. Clinton acknowledged, but "because it does far more good than harm.↑" Despite resistance from some lawmakers I his own party, the president predicted the bill would pass.

Rule 4 identifies an *edu* boundary before the occurrence of an "and" followed by a verb in the past tense (VPT). This rule will correctly identify the marked *edu* boundary in sentence 4.5.

(4.5) Ashley Boone ran marketing and distribution ↑ and left the company late last year.

Rule 5 inserts *edu* boundaries before the occurrence of the word "until", provided that "until" is followed not necessarily by a verb. This rule will correctly insert an *edu* boundary in example 4.6.

(4.6) Several appointees of President Bush are likely to stay in office at least temporarily,↑ until permanent successors can be names.

Rule 6 is an automatically derived rule that mirrors the manually derived rule specific to COMMA-like actions in the

surface-based unit identification algorithm. Rule 6 will correctly insert an *edu* boundary after the comma marked in example 4.7, because the marker "While" was used at the beginning of the sentence.

(4.7) While the company hasn't commented on the probe,↑ persons close to the board said that Messrs. Lavin and Young, along with some other top Woolworth executives were under investigation by the special committee for their possible involvement in the alleged irregularities.

Rule 7 specifies that no elementary or parenthetical unit boundary should be inserted immediately before a DOT.

As one can notice, the rules in Figure 6a are more complex than typical manually derived rules. The automatically derived rules make use not only of orthographic and cue-phrase-specific information, but also of syntactic information, which is encoded as part of speech tags.

In step 606, the C4.5 program was used in order to learn decision trees and rules that classify lexemes as boundaries of sentences, *edus*, or parenthetical units, or as non-boundaries. Learning was accomplished both from binary representations (when possible) and non-binary representations of the cases. (Learning from binary representations of features in the Brown corpus was too computationally expensive to terminate - the Brown data file had about 0.5 Giga-bytes.) In general the binary representations yielded slightly better results than the non-binary

representations and the tree classifiers were slightly better than the rule-based ones.

Table 1 shows accuracy results of non-binary, decision-tree classifiers. The accuracy figures were computed using a ten-fold cross-validation procedure. In Table 1, B1 corresponds to a majority-based baseline classifier that assigns the class "none" to all lexemes, and B2 to a baseline classifier that assigns a sentence boundary to every "DOT" (that is, a period (.), question mark (?), and/or exclamation point (!)) lexeme and a non-boundary to all other lexemes.

Corpus	# cases	B1(%)	B2(%)	Acc(%)
MUC	14362	91.28	93.1	96.24±0.06
WSJ	31309	92.39	94.6	97.14±0.10
Brown	72092	93.84	69.8	97.87±0.04

Table 1: Performance of a discourse segmenter that uses a decision-tree, non-binary classifier.

Figure 7 shows the learning curve that corresponds to the MUC corpus. It suggests that more data can increase the accuracy of the classifier.

The confusion matrix shown in Table 2 corresponds to a non-binary-based tree classifier that was trained on cases derived from 27 Brown texts and that was tested on cases derived from 3 different Brown texts, which were selected randomly. The matrix shows that the segmenter encountered some difficulty with

identifying the beginning of parenthetical units and the intra-sentential *edu* boundaries; for example, it correctly identifies 133 of the 220 *edu* boundaries. The performance is high with respect to recognizing sentence boundaries and ends of parenthetical units. The performance with respect to identifying sentence boundaries appears to be close to that of systems aimed at identifying "only" sentence boundaries, such as described in David D. Palmer and Marti A. Hearst, "Adaptive multilingual sentence boundary disambiguation," Computational Linguistics, 23(2):241–269 (1997) (hereinafter, "Hearst (1997)"), whose accuracy is in the range of 99%.

Action		(a)	(b)	(c)	(d)	(e)
sentence-break	(a)	272				4
edu-break	(b)		133		3	84
start-paren	(c)			4		26
end-paren	(d)				20	6
none	(e)	2	38	1	4	7555

Table 2: Confusion matrix for the decision-tree, non-binary classifier (the Brown corpus).

Training the Discourse Parser

Figure 8 shows a generalized flowchart for a process 800 for generating decision rules for the discourse parser. Put another way, the process 800 can be used to train the discourse parser about when and under what circumstances, and in what sequence, it should perform the various shift-reduce operations.

In step 802, the process receives as input the training corpus of discourse trees and, for each discourse tree, a set of *edus* from the discourse segmenter. Next, in step 804, for each discourse tree / *edu* set, the process 800 determines a sequence of shift-reduce operations that reconstructs the discourse tree from the *edus* in that tree's corresponding set. Next, in step 806, the process 800 associates features with each entry in each sequence. Finally, in step 808, the process 800 applies a learning algorithm (e.g., C4.5) to generate decision rules 810 for the discourse parser. As noted above, the discourse parser will then be able to use these decision rules 810 to determine the rhetorical structure for any input text and, from it, generate a discourse tree as output.

Additional details of training the discourse parser follow.

Shift-Reduce Action Identifier: Generation of learning examples

The learning cases were generated automatically, in the style of Magerman, "Statistical decision-tree models for parsing," *Proceedings of ACL'95*, pages 276-283 (1995), by traversing in-order the final rhetorical structures built by annotators and by generating a sequence of discourse parse actions that used only SHIFT and REDUCE operations of the kinds discussed above. When a derived sequence is applied as described above with respect to the parsing model, it produces a rhetorical

tree that is a one-to-one copy of the original tree that was used to generate the sequence. For example, the tree at the bottom of Figure 4 - the tree found at the top of the stack at step *i* + 4 - can be built if the following sequence of operations is performed: {SHIFT 12; SHIFT 13; REDUCE-ATTRIBUTION-NS; SHIFT 14; REDUCE-JOINT-NN; SHIFT 15; REDUCE-CONTRAST-SN; SHIFT 16; SHIFT 17; REDUCE-CONTRAST-SN; SHIFT 18; SHIFT 19; REDUCE-APPOSITION-NS; REDUCE ATTRIBUTION-NS; REDUCE-ELABORATION-NS.}

The Shift-Reduce Action Identifier: Features used for learning

To make decisions with respect to parsing actions, the shift-reduce action identifier focuses on the three topmost trees in the stack and the first *edt* in the input list. These trees are referred to as the trees "in focus." The identifier relies on the following classes of features: structural features, lexical (cue-phase-like) features, operational features, and semantic-similarity-based features. Each is described in turn.

Structural features.

Structural features include the following:

(1) Features that reflect the number of trees in the stack and the number of *edts* in the input list.

(2) Features that describe the structure of the trees in focus in terms of the type of textual units that they subsume

(sentences, paragraphs, titles). These may include the number of immediate children of the root nodes, the rhetorical relations that link the immediate children of the root nodes, , and the like. The identifier assumes that each sentence break that ends in a period and is followed by two '\n' characters, for example, is a paragraph break; and that a sentence break that does not end in a punctuation mark and is followed by two '\n' characters is a title.

Lexical (cue-phrase-like) and syntactic features.

Lexical features include the following:

(1) Features that denote the actual words and POS tags of the first and last two lexemes of the text spans subsumed by the trees in focus.

(2) Features that denote whether the first and last units of the trees in focus contain potential discourse markers and the position of these markers in the corresponding textual units (beginning, middle, or end).

Operational features.

Operational features includes features that specify what the last five parsing operations performed by the parser were. These features could be generated because, for learning, sequences of shift-reduce operations were used and not discourse trees.

Semantic-similarity-based features.

Semantic-similarity-based features include the following:

(1) Features that denote the semantic similarity between the textual segments subsumed by the trees in focus. This similarity is computed by applying in the style of Hearst (1997) a cosine-based metric on the morphed segments. If two segments S_1 and S_2 are represented as sequences of $(t, w(t))$ pairs, where t is a token and $w(t)$ is its weight, the similarity between the segments can be computed using the formula shown below, where $w(t)_{S_1}$ and $w(t)_{S_2}$ represent the weights of token t in segments S_1 and S_2 respectively.

$$sim(S_1, S_2) = \frac{\sum_{t \in S_1 \cup S_2} w(t)_{S_1} w(t)_{S_2}}{\sqrt{\sum_{t \in S_1} w(t)_{S_1}^2 \sum_{t \in S_2} w(t)_{S_2}^2}}$$

The weights of tokens are given by their frequencies in the segments.

(2) Features that denote Wordnet-based measures of similarity between the bags of words in the promotion sets of the trees in focus. Fourteen Wordnet-based measures of similarity were used, one for each Wordnet relation (Fellbaum, *Wordnet: An Electronic Lexical Database*, The MIT Press, 1998). Each of these similarities is computed using a metric similar to the cosine-based metric. Wordnet-based similarities reflect the degree of

synonymy, antonymy, meronymy, hyponymy, and the like between the textual segments subsumed by the trees in focus. The Wordnet-based similarities are computed over the tokens that are found in the promotion units associated with each segment. If the words in the promotion units of two segments S_1 and S_2 are represented as two sequences W_1 and W_2, the Wordnet-based similarities between the two segments can be computed using the formula shown in below, where the function $\sigma_{R(w1, w2)}$ returns 1 if there exists a Wordnet relation of type R between the words w_1 and w_2, and 0 otherwise.

$$sim_{wordnet\,Relation}(W_1, W_2) = \frac{\sum_{w_1 \in W_1, w_2 \in W_2} \sigma_{wordnet\,Relation(w_1, w_2)}}{|W_1| x |W_2|}$$

The Wordnet-based similarity function takes values in the interval [0,1]: the larger the value, the more similar with respect to a given Wordnet relation the two segments are.

In addition to these features that modeled the Wordnet-based similarities of the trees in focus, 14 X 13/2 = 91 *relative* Wordnet-based measures of similarity were used, one of each possible pair of Wordnet-based relations. For each pair of Wordnet-based measures of similarity w_{r1} and w_{r2}, each relative measure (feature) takes the value <, =, or >, depending on whether the Wordnet-based similarity w_{r1} between the bags of words in the promotion sets of the trees in focus is lower, equal, or higher than the Wordnet-based similarity w_{r2} between

the same bags of words. For example, if both the synonymy- and meronymy-based measures of similarity are 0, the relative similarity between the synonymy and meronymy of the trees in focus will have the value =.

A binary representation of these features yields learning examples with 2789 features/example.

Examples of Rule Specific to the Action Identifier

Figure 8A shows some of the rules that were learned by the C4.5 program using a binary representation of the features and learning cases extracted from the MUC corpus. Rule 1, which is similar to a typical rule derived manually, specifies that if the last lexeme in the tree at position *top* - 1 in the stack is a comma and there is a marker "if" that occurs at the beginning of the text that corresponds to the same tree, then the trees at position *top* - 1 and *top* should be reduced using a REDUCE-CONDITION-SN operation. This operation will make the tree at position *top* - 1 the satellite of the tree at position *top*. If the *edt* at position *top* - 1 in the stack subsumes unit 1 in example 5.1 and the *edt* at position *top* subsumes unit 2, this reduce action will correctly replace the two *edts* with a new rhetorical tree, that shown in Figure 8B.

(5.1) [If you refer to someone as a butt-head,[1]][ordinarily speaking, no one is going to take that as any specific charge of any improper conduct or insinuation of any character trait.[2]]

Rule 2 makes the tree at the *top* of the stack the BACKGROUND-CIRCUMSTANCE satellite of the tree at position *top* - 1 when the first word in the text subsumed by the *top* tree is "when", which is a while-adverb (WRB), when the second word in the same text is not a gerund or past participle verb (VBG), and when the cosine-based similarity between the text subsumed by the top node in the stack and the first unit in the list of elementary discourse units that have not been shifted to the stack is greater than 0.0793052. If the *edt* as position *top* - 1 in the stack subsumes unit 1 in example 5.2 and the *edt* at position *top* subsumes unit 2, rule 2 will correctly replace the two *edts* with the rhetorical tree shown in Figure 8C.

(5.2) [Mrs. Graham, 76 years old, has not been involved in day-to-day operations at the company since May 1991.[1]][when Mr. Graham assumed the chief executive officer's title.[2]]

In case the last word in the text subsumed by the tree at position *top* - 1 in the stack is a plural noun (NNS), the first word in the text subsumed by the tree at the *top* of the stack is a preposition or subordinating conjunction (IN), and the hyponymy-based similarity between the two trees at the top of the stack is equal with their synonymy-based similarity, then the

action to be applied is REDUCE-BACKGROUND-CIRCUMSTANCE-NS. When this rule is applied in conjunction with the *edts* that correspond to the units marked in 5.3, the resulting tree has the same shape as the tree shown in Figure 8C.

(5.3) [In an April 7 *Wall Street Journal* article, several experts suggested that IBM's accounting grew much more liberal since the mid-1980s[1]][as its business turned sour.[2]]

When the tree at the top of the stack subsumes a paragraph and starts with the marker "but", the action to be applied is REDUCE-CONTRAST-NN. For example, if the trees at the top of the stack subsume the paragraphs shown in Figure 8D and are characterized by promotion sets P1 and P2, as a result of applying rule 4 in Figure 8A, one would obtain a new tree, whose shape is shown in Figure 8E; the promotion units of the root node of this tree are given by the union of the promotion units of the child nodes.

The last rule in Figure 8A reflects the fact that each text in the MUC corpus is characterized by a title. When there are no units left in the input list (noUnitsInList = 0) and a tree that subsumes the whole text has been built (noTreesInStack <=2), the two trees that are left in the tree - the one that corresponds to the title and the one that corresponds to the text - are reduced using a REDUCE-TEXTUAL-NN operation.

Evaluation of shift-reduce-action identifier

Table 3 below displays the accuracy of the shift-reduce action identifiers, determined for each of the three corpora (MUC, Brown, WSJ) by means of a ten-fold cross-validation procedure. In table 3, the B3 column gives the accuracy of a majority-based classifier, which chooses action SHIFT in all cases. Since choosing only the action SHIFT never produces a discourse tree, column B4 presents the accuracy of a baseline classifier that chooses shift-reduce operations randomly, with probabilities that reflect the probability distribution of the operations in each corpus.

Corpus	# cases	B3(%)	B4(%)	Acc(%)
MUC	1996	50.75	26.9	61.12±1.61
WSJ	4360	50.34	27.3	61.65±0.41
Brown	8242	50.18	28.1	61.81±0.48

Table 3: Performance of the tree-based, shift-reduce action classifiers.

Figure 9 shows the learning curve that corresponds to the MUC corpus. As in the case of the discourse segmenter, this learning curve also suggests that more data can increase the accuracy of the shift-reduce action identifier.

Evaluation of the rhetorical parser

By applying the two classifiers sequentially, one can derive the rhetorical structure of any text. The performance results

presented above suggest how well the discourse segmenter and the shift-reduce action identifier perform with respect to individual cases, but provide no information about the performance of a rhetorical parser that relies on these classifiers.

Corpus	Segmenter	Training corpus	Elementary units				Hierarchical spans				Span nuclearity				Rhetorical relations			
			Judges		Parser		Judges		Parser		Judges		Parser		Judges		Parser	
			R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P
MUC	DT	MUC	88.0	88.0		100.0	84.4	84 4	38.2	61.0	79.1	83.5	25 5	51.5	78.6	78.6	14 9	28 7
	DT	All			37.1				70.9	72.8			58.3	68.9			38.4	45 3
	M	MUC				96.9			87.5	82.3			68.8	78.2			72.4	62.8
	M	All			75 4	100.0			84.8	73.5			71.0	69 3			66.5	53.9
					100.0	100.0												
					100 0													
WSJ	DT	WSJ	85 1	86.8			79 9	80.1	34.0	65 8	67.6	77.1	21.6	54.0	73.1	73.3	13.0	34.3
	DT	All			18 1	95.8			40.1	66.3			30 3	58.5			17.3	36.0
	M	WSJ							83.4	84.2			63 7	79.9			56.3	57 9
	M	All			25.1	79.6			83.0	85.0			69.0	82.4			59.8	63.2
					100.0	100 0												
					100 0	100 0												
Brown	DT	Brown	89.5	88 5			80 6	79 5	57.3	63 3	67.6	75.8	44.6	57.3	69 7	68.3	26 7	35 3
	DT	All			60 5	79 4			44.7	59.1			33 2	51 8			15.7	25.7
	M	Brown							88 1	73.4			60.1	67 0			59 5	45 5
	M	All			44 2	80.3			80 8	77.5			60.0	72 0			51 8	44 7
					100.0	100 0												
					100 0	100 0												

Table 4: Performance of the rhetorical parser: labeled (R)ecall and (P)recision. The segmenter is either Decision-Tree-Based (DT) or Manual (M).

In order to evaluate the rhetorical parser as a whole, each corpus was partitioned randomly into two sets of texts: 27 texts were used for training and the last 3 texts were used for testing. The evaluation employs "labeled recall" and "labeled precision" measures, which are extensively used to study the performance of syntactic parsers. "Labeled recall" reflects the number of correctly labeled constituents identified by the rhetorical parser with respect to the number of labeled

constituents in the corresponding manually built tree. "Labeled precision" reflects the number of correctly labeled constituents identified by the rhetorical parser with respect to the total number of labeled constituents identified by the parser.

Labeled recall and precision figures were computed with respect to the ability of the discourse parser to identify elementary units, hierarchical text spans, text span nuclei and satellites, and rhetorical relations. Table 4 displays results obtained using segmenters and shift-reduce action identifiers that were trained either on 27 texts from each corpus and tested on 3 unseen texts from the same corpus; or that were trained on 27 X 3 texts from all corpora and tested on 3 unseen texts from each corpus. The training and test texts were chosen randomly. Table 4 also displays results obtained using a manual discourse segmenter, which identified correctly all *edus*. Since all texts in the corpora were manually annotated by multiple judges, an upper-bound of the performance of the rhetorical parser was computed by calculating, for each text in the test corpus and each judge, the average labeled recall and precision figures with respect to the discourse trees built by the other judges. Table 4 displays these upper-bound figures as well.

The results in table 4 primarily show that errors in the discourse segmentation stage affect significantly the quality of the trees the parser builds. When a segmenter is trained only on

27 texts (especially for the MUC and WSJ corpora, which have shorter texts that the Brown corpus), it has very low performance. Many of the intra-sentential *edu* boundaries are not identified, and as a consequence, the overall performance of the parser is low. When the segmenter is trained on 27 X 3 texts, its performance increases significantly with respect to the MUC and WSJ corpora, but decreases with respect to the Brown corpus. This can be explained by the significant differences in style and discourse marker usage between the three corpora. When a perfect segmenter is used, the rhetorical parser determines hierarchical constituents and assigns them a nuclearity status at levels of performance that are not far from those of humans. However, the rhetorical labeling of discourse spans even in this case is about 15-20% below human performance. These results suggest that the features used are sufficient for determining the hierarchical structure of texts and the nuclearity statuses of discourse segments.

Alternative embodiments of the discourse parser and its parsing procedure are possible. For example, probabilities could be incorporated into the process that builds the discourse trees. Alternatively, or in addition, multiple trees could be derived in parallel and the best one selected in the end. In the current embodiment, the final discourse tree is generated in a sequence of deterministic steps with no recursion or branching.

Alternatively, it is possible to associate a probability with each individual step and build the discourse tree of a text by exploring multiple alternatives at the same time. The probability of a discourse tree is given by the product of the probabilities of all steps that led to the derivation of that tree. In such a case, the discourse tree of a text will be taken to be the resulting tree of maximum probability. An advantage of such an approach is that it enables the creation of multiple trees, each one having associated a probability.

SUMMARIZATION

Various summarizing systems and techniques are described in detail below. In general, two different embodiments of a summarizer are described. First, a "channel-based" summarizer that uses a probabilistic approach for summarization (equivalently, compression) is described, and second, a "decision-based" summarizer that uses learned decision rules for summarization is described.

Channel-Based Summarizer

This section describes a probabilistic approach to the compression problem. In particular, a *"noisy channel"* framework is used. In this framework, a long text string is regarded as (1) originally being a short string, that (2) someone added some

additional, optional text to it. Compression is a matter of identifying the original short string. It is not critical whether or not the "original" string is real or hypothetical. For example, in statistical machine translation, a French string could be regarded as originally being in English, but having noise added to it. The French may or may not have been translated from English originally, but by removing the noise, one can hypothesize an English source - and thereby translate the string. In the case of compression, the noise consists of optional text material that pads out the core signal. For the larger case of text summarization, it may be useful to imagine a scenario in which a news editor composes a short document, hands it to a reporter, and tells the reporter to "flesh it out" . . . which results in the final article published in the newspaper. In summarizing the final article, the summarizer typically will not have access to the editor's original version (which may or may not exist), but the summarizer can guess at it - which is where probabilities come in.

In a noisy channel application, three problems must be solved:

- Source model. To every string s a probability $P(s)$ must be assigned. $P(s)$ represents the chance that s is generated as an "original short string" in the above hypothetical process. For example, it may be desirable to have $P(s)$ to be very low if s is ungrammatical.

- Channel model. To every pair of strings (s,t) a probability $P(t \mid s)$ is assigned. $P(t \mid s)$ represents the chance that when the short string s is expanded, the result is the long string t. For example, if t is the same as s except for the extra word "not," then it may be desirable to have a very low $P(t \mid s)$ because the word "not" is not optional, additional material.

- Decoder. Given a long string t, a short string s is searched for that maximizes $P(s \mid t)$. This is equivalent to searching for the s that maximizes $P(s) \cdot P(t \mid s)$.

It is advantageous to break down the noisy channel problem this way, as it decouples the somewhat independent goals of creating a short text that (1) is grammatical and coherent, and (2) preserves important information. It is easier to build a channel model that focuses exclusively on the latter, without having to worry about the former. That is, one can specify that a certain substring may represent unimportant information without worrying that deleting the substring will result in an ungrammatical structure. That concern is left to the source model, which worries exclusively about well-formedness. In that regard, well-known prior work in source language modeling for speech recognition, machine translation, and natural language generation can be used. The same goes for actual compression ("decoding" in noisy-channel jargon) - one can re-use generic software packages to solve problems in all these application domains.

Statistical Models

In the experiments discussed here, relatively simple source and channel models were built and used. In a departure from the above discussion and from previous work on statistical channel models, probabilities $P_{tree}(s)$ and $P_{expand\_tree}(t \mid s)$ were assigned to trees rather than strings. In decoding a new string, first it is parsed into a large syntactic tree t (for example, using the parser described in M. Collins, "Three generative, lexicalized models for statistical parsing," *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, 16-23 (1997)) and then various small syntactic trees are hypothesized and ranked.

Good source strings are ones that have both (1) a normal-looking parse tree, and (2) normal-looking word pairs. $P_{tree}(s)$ is a combination of a standard probabilistic context-free grammar (PCFG) score, which is computed over the grammar rules that yielded the tree s, and a standard word-bigram score, which is computed over the leaves of the tree. For example, the tree s = (S (NP John) (VP (VB saw) (NP Mary))) is assigned a score based on these factors:

$$
\begin{aligned}
P_{tree}(s) = {} & P(\text{TOP} \rightarrow \text{S} \mid \text{TOP}) \cdot \\
& P(\text{S} \rightarrow \text{NP VP} \mid \text{S}) \cdot P(\text{NP} \rightarrow \text{John} \mid \text{NP}) \cdot \\
& P(\text{VP} \rightarrow \text{VP NP} \mid \text{VP}) \cdot P(\text{VP} \rightarrow \text{saw} \mid \text{VB}) \cdot \\
& P(\text{NP} \rightarrow \text{Mary} \mid \text{NP}) \cdot \\
& P(\text{John} \mid \text{EOS}) \cdot P(\text{saw} \mid \text{John}) \cdot \\
& P(\text{Mary} \mid \text{saw}) \cdot P(\text{EOS} \mid \text{Mary})
\end{aligned}
$$

The stochastic channel model performs minimal operations on a small tree *s* to create a larger tree *t*. For each internal node in *s*, an *expansion template* is chosen probabilistically based on the labels of the node and its children. For example, when processing the S node in the tree above, one may wish to add a prepositional phrase as a third child. This is done with probability P(S → NP VP PP | S → NP VP). Or one may choose to leave it alone, with probability P(S → NP VP | S → NP VP). After an expansion template is chosen, then for each new child node introduced (if any), a new subtree is grown rooted at that node-for example, (PP (P in) (NP Pittsburgh)). Any particular subtree is grown with probability given by its PCFG factorization, as above (no bigrams).

Example of using statistical models for compression

This example demonstrates how to tell whether one potential compression is more likely than another, according to the statistical models described above. Figure 11 shows examples of parse trees. As shown, the tree *t* in Figure 11 spans the string **abcde**. Consider the parse tree for compression *s*1, which is also shown in Figure 11.

The factors $P_{tree}(s1)$ and $P_{expand\_tree}(t \mid s1)$ are computed. Breaking this down further, the source PCFG and word-bigram factors, which describe $P_{tree}(s1)$, are as follows:

P(TOP → G | TOP)	P(H → a | H)
P(G → H A | G)	P(C → b | C)
P(A → C D | A)	P(D → e |D)
P(a | EOS)	P(e | b)
P(b | a)	P(EOS | e)

The channel expansion-template factors and the channel PCFG (new tree growth) factors, which describe $P_{expand\_tree}(t \mid s1)$ are:

P(G → H A | G → H A)	
P(A → C B D | A → C D)	
P(B → Q R | B)	P(Z → c | Z)
P(Q → Z | Q)	P(R → d | R)

A different compression will be scored with a different set of factors. For example, consider a compression of t that leaves t completely untouched. In that case, the source costs $P_{tree}(t)$ are:

P(TOP → G | TOP)	P(H → a | H)	P(a | EOS)
P(G → H A | G)	P(C → b | C)	P(b | a)
P(A → C D | A)	P(Z → c | Z)	P(c | b)
P(B → Q R | B)	P(R → d |R)	P(d | c)
P(Q → Z | Q)	P(D → e | D)	P(e | d)
		P(EOS | e)

The channel costs $P_{expand\_tree}(t \mid t)$ are:

P(G → H A | G → H A)
P(A → C B D | A → C B D)
P(B → Q R | B → Q R)
P(Q → Z | Q → Z)

Now, the following values are compared - $P_{expand\_tree}(s1 \mid t) = P_{tree}(s1) \cdot P_{expand\_tree}(t \mid s1))/P_{tree}(t)$ versus $P_{expand\_tree}(t \mid t) = P_{tree}(t) \cdot P_{expand\_tree}(t \mid t) / P_{tree}(t)$ - and the more likely one is selected. Note that $P_{tree}(t)$ and all the PCFG factors can be canceled out, as they appear in any potential compression. Therefore, one need only compare compressions of the basis of the expansion-template probabilities and the word-bigram probabilities. The quantities that differ between the two proposed compressions are boxed above. Therefore, $s1$ will be preferred over t if and only if:

$$P(E \mid b) \cdot P(A \rightarrow C\ B\ D \mid A \rightarrow C\ D) >$$
$$P(b \mid a) \cdot P(c \mid b) \cdot P(d \mid c) \cdot$$
$$P(A \rightarrow C\ B\ D \mid A \rightarrow C\ B\ D) \cdot$$
$$P(B \rightarrow Q\ R \mid B \rightarrow Q\ R) \cdot P(Q \rightarrow Z \mid Q \rightarrow Z)$$

Training Corpus

In order to train the channel-based summarizing system, the Ziff-Davis corpus - a collection of newspaper articles announcing computer products - was used. Many of the articles in the corpus are paired with human written abstracts. A set of 1067 sentence pairs were automatically extracted from the corpus. Each pair consisted of a sentence $t = t_1, t_2, \ldots, t_n$ that occurred in the article and a possible compressed version of it $s = s1, s_2, \ldots s_m$, which occurred in the human written abstract. Figure 12 shows a few examples of sentence pairs extracted from the corpus.

This corpus was chosen because it is consistent with two desiderata specific to summarization work: (i) the human-written Abstract sentences are grammatical; (ii) the Abstract sentences represent in a compressed form the salient points of the original newspaper Sentences. The uncompressed sentences were kept in the corpus as well, since an objective was to learn not only *how* to compress a sentence, but also *when* to do it.

Learning Model Parameters For Channel-Based Summarizer

Expansion-template probabilities were collected from parallel corpus. First, both sides of the parallel corpus were parsed, and then corresponding syntactic nodes were identified. For example, the parse tree for one sentence may begin

```
(S (NP . . . )
   (VP . . . )
   (PP . . . ))
```

while the parse tree for its compressed version may begin

```
(S (NP . . . )
   (VP . . . )).
```

If these two S nodes are deemed to correspond, then a joint event (S → NP VP, S → NP VP PP) is recorded. Afterwards the events are normalized so that the probabilities add up to one. Not all nodes have corresponding partners; some non-correspondences are due to incorrect parses, while others are due

to legitimate reformulations that are beyond the scope of the simple channel model. Standard methods based on counting and parameters were used to estimate word-bigram probabilities.

Decoding

A vast number of potential compressions of a large tree *t* exist, but all of them can be packed efficiently into a shared-forest structure. For each node of *t* that has *n* children, the following operations are performed:

- generate $2^n - 1$ new nodes, one for each non-empty subset of the children, and
- Pack those nodes so that they are referred to as a whole.

For example, consider the large tree t above. All compressions can be represented with the following forest:

G → H A	B → R	A → B C	H → a
G → H	Q → Z	A → C	C → b
G → A	A → C B D	A → B	Z → c
B → Q R	A → C B	A → D	R → d
B → Q	A → C D		D →e

An expansion-template probability can be assigned to each node in the forest. For example, to the B → Q node, one can assign P(B → Q R | B → Q). If the observed probability from the parallel corpus is zero, then a small floor value of 10^{-6} is assigned. In reality, forests are produced that are much slimmer, as only methods of compressing a node that are locally

grammatical according to the Penn Treebank are considered. (Penn Treebank is a collection of manually built syntactic parse trees available from the Linguistic Data Consortium at the University of Pennsylvania.) If a rule of the type A → C B has never been observed, then it will not appear in the forest.

Next, a set of high-scoring trees is extracted from the forest, taking into account both expansion-template probabilities and word-bigram probabilities. A generic extractor such as described by I. Langkilde, "Forest-based statistical sentence generation," *Proceedings of the 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistics* (2000) can be used for this purpose.

The extractor selects the trees with the best combination of word-bigram and expansion template scores. It returns a list of such trees, one for each possible compression length. For example, as shown in Figure 16, for the sentence *Beyond that basic level, the operations of the three products vary*, the following "best" compressions are obtained, with negative log-probabilities shown in parentheses (smaller = more likely):

Length Selection

It is useful to have multiple answers to choose from, as one user may seek 20% compression, while another seeks 60% compression. However, for purposes of evaluation, the

summarizing system was designed to be able to select a single compression. If log-probabilities as shown in Figure 16 are relied upon, then typically the shortest compression will be chosen. (Note above, however, how the three-word compression scores better than the two-word compression, as the models are not entirely happy removing the article "the"). To create a more reasonable competition, the log-probability is divided by the length of the compression, rewarding longer strings. This technique often is applied speech recognition.

If this normalized score is plotted against compression length, typically a (bumpy) U-shaped curve results, as illustrated in Figure 13. In a typical more difficult case, a 25-word sentence may be optimally compressed by a 17-word version. Of course, if a user requires a shorter compression than that, another region of the curve may be selected and inspected for a local minimum.

Figs. 17 and 18 are respectively generalized flowcharts of the channel-based summarization and training processes described above.

As shown in Fig. 17, the first step 1702 in the channel-based summarization process 1700 is to receive the input text. Although the embodiment described above uses sentences as the input text, any other text segment could be used instead, for example, clauses, paragraphs, or entire treatises.

Next, in step 1704, the input text is parsed to produce a syntactic tree in the style of Figure 11, which is used in step 1706 as the basis of generating multiple possible solutions (e.g., the shared-forest structure described above). If a whole text is given as input, the text can be parsed to produce a discourse tree, and the algorithm described here will operate on the discourse tree.

Next, the multiple possible solutions generated in step 1706 are ranked using pre-generated ranking statistics from a statistical model. For example, step 1706 may involve assigning an expansion-template probability to each node in the forest, as described above.

Finally, the best scoring candidate (or candidates) is (are) chosen as the final compression solution(s) in step 1710. As described above, the best scoring candidate may be the one having the smallest log-probability / length of compression ratio.

Fig. 18 shows a generalized process for training a channel-based summarizer. As shown therein, the process 1800 starts in step 1802 with an input training set (or corpus). As discussed above, this input training set comprises pairs of long-short text fragments, for example, long/short sentence pairs or treatise/abstract pairs. Typically, because a main purpose of the training set is to teach the summarizer how to properly compress text, the training set used will have been generated

manually by experienced editors who know how to create relevant, coherent and grammatical summaries of longer text segments.

Next, in step 1804, the long-short text pairs are parsed to generate syntactic parse trees such as shown in Figure 11, thereby resulting in corresponding long-short syntactic tree pairs. Each item of text in each pair is parsed individually in this manner. Also, the entire text is parsed using the discourse parser.

Next, in step 1806, the resulting parse tree pairs are compared - that is, the discourse or syntactic parse tree for a long segment is compared against the discourse or syntactic parse tree for its paired short segment - to identify similarities and differences between nodes of the tree pairs. A difference might occur, for example, if, in generating the short segment, an editor deleted a prepositional phrase from the long segment. In any event, the results of this comparison are "events" that are collected for each of the long/short pairs and stored in a database. In general, two different types of events are detected: "joint events" which represent a detected correspondence between a long and short segment pair and Context-Free Grammar (CFG) events, which relate only to characteristics of the short segment in each pair.

Next, in step 1808, the collected events are normalized to generate probabilities. These normalized events collectively

represent the statistical learning model 1810 used by the channel-based summarizer.

Decision-based Summarizer

A description of a decision-based, history model of sentence compression follows. As in the noisy-channel approach, it is assumed that a parse tree *t* is given as input. The goal is to "rewrite" *t* into a smaller tree *s*, which corresponds to a compressed version of the original sentence subsumed by *t*. Assume the trees *t* and *s2* in Figure 11 are in the corpus. In the decision-based summarizer model, the question presented is how may tree *t* be rewritten into *s2*. One possible solution is to decompose the rewriting operation into a sequence of shift-reduce-drop actions that are specific to an extended shift-reduce parsing paradigm.

In the decision-based model, the rewriting process starts with an empty Stack and an Input List that contains the sequence of words subsumed by the large tree *t*. Each word in the input list is labeled with the name of all syntactic constituents in *t* that start with that word (see Figure 14). At each step, the rewriting module applies an operation that is aimed at reconstructing the smaller tree *s2*. In the context of the sentence-compression module, four types of operations are used:

- SHIFT operations transfer the first word from the input list into the stack;

- REDUCE operations pop the *k* syntactic trees located at the top of the stack; combine them into a new tree; and push the new tree on the top of the stack. Reduce operations are used to derive the structure of the syntactic tree of the short sentence.

- DROP operations are used to delete from the input list subsequences of words that correspond to syntactic constituents. A DROP X operations deletes from the input list all words that are spanned by constituent X in *t*.

- ASSIGNTYPE operations are used to change the label of trees at the top of the stack. These actions assign POS tags to the words in the compressed sentence, which may be different from the POS tags in the original sentence.

The decision-based model is more flexible than the channel model because it enables the derivation of a tree whose skeleton can differ quite drastically from that of the tree given as input. For example, the channel-based model was unable to obtain tree *s*2 from *t*. However, using the four operations listed above (SHIFT, REDUCE, DROP, ASSIGNTYPE), the decision-based model was able to rewrite a tree *t* into *any* tree *s*, as long as an in-order traversal of the leaves of *s* produces a sequence of words that occur in the same order as the words in the tree *t*. For example, the tree *s*2 can be obtained from the tree *t* by following this sequence of actions, whose effects are shown in Figure 14: SHIFT, ASSIGNTYPE H; SHIFT; ASSIGNTYPE K; REDUCE 2 F; DROP B; SHIFT; ASSIGNTYPE D; REDUCE 2 G.

To save space, the SHIFT and ASSIGNTYPE operations are shown in Fig. 14 on the same line. However, it should be understood that the SHIFT and ASSIGNTYPE operations correspond to two distinct actions. The ASSIGNTYPE K operation rewrites the POS tag of the word b; the REDUCE operations modify the skeleton of the tree given as input. To increase readability, the input list is shown in Fig. 14 in a format that resembles the graphical representation of the trees in Figure 11.

Learning the Parameters of (Training) the Decision-Based Model

To train the decision-based model, each configuration of our shift-reduce-drop rewriting model is associated with a learning case. The learning cases are generated automatically by a program that derives sequences of actions that map each of the large trees in our corpus into smaller trees. The rewriting procedure simulates a bottom-up reconstruction of the smaller trees.

Overall, the 1067 pairs of long and short sentences yielded 46383 learning cases. Each case was labeled with one action name from a set of 210 possible actions: There are 37 different ASSIGNTYPE actions, one for each POS tag. There are 63 distinct DROP actions, one for each type of syntactic constituent that can be deleted during compression. There are 109 distinct REDUCE actions, one for each type of reduce operation that is applied

during the reconstruction of the compressed sentence. And there is one SHIFT operation. Given a tree *t* and an arbitrary configuration of the stack and input list, the purpose of the decision-based classifier is to learn what action to choose from the set of 210 possible actions.

To each learning example, a set of 99 features was associated from the following two classes: operation features and original-tree-specific features.

Operational features reflect the number of trees in the stack, the input list, and the types of the last five operations performed. Operational features also encode information that denotes the syntactic category of the root nodes of the partial trees built up to a certain time. Examples of operational features include the following: numberTreesInStack, wasPreviousOperationShift, syntacticLabelOfTreeAtTheTopOfStack.

Original-tree-specific features denote the syntactic constituents that start with the first unit in the input list. Examples of such features include inputListStartsWithA_CC and inputListStartsWithA_PP.

The decision-based compression module uses the C4.5 program as described in J. Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufmann Publishers (1993), in order to learn decision trees that specify how large syntactic trees can be compressed into shorter trees. A ten-fold cross-validation

evaluation of the classifier yielded an accuracy of 98.16% (±0.14). A majority baseline classifier that chooses the action SHIFT has an accuracy of 28.72%.

Figure 18A shows examples of rules that were learned automatically by the C4.5 program. As seen therein, Rule 1 enables the deletion of WH prepositional phrases in the context in which they follow other constituents that the program decided to delete. Rule 2 enables the deletion of WHNP constituents. Since this deletion is carried out only when the stack contains only one NP constituent, it follows that this rule is applied only in conjunction with complex nounphrases that occur at the beginning of sentences. Rule 3 enables the deletion of adjectival phrases.

Employing the decision-based model

To compress sentences, the shift-reduce-drop model is applied in a deterministic fashion. The sentence to be compressed is parsed and the input list is initialized with the words in the sentence and the syntactic constituents that "begin" at each word, as shown in Figure 14. Afterwards, the learned classifier is asked in a stepwise manner what action to propose. Each action is then simulated, thus incrementally building a parse tree. The procedure ends when the input list is empty and when the stack contains only one tree. An in-order traversal of

the leaves of this tree produces the compressed version of the sentence given as input.

Because the decision-based model is deterministic, it produces only one output. An advantage of this result is that compression using the decision-based model is very fast: it takes only a few milliseconds per sentence. One potential disadvantage, depending on one's objectives, is that the decision-based model does not produce a range of compressions, from which another system may subsequently choose. It would be relatively straightforward to extend the model within a probabilistic framework by applying, for example, techniques described in D. Magerman, "Statistical decision-tree models for parsing," Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, 276-283 (1995).

Figs. 19 and 20 are respectively generalized flowcharts of the decision-based summarization and training processes described above.

As shown in Fig. 19, the first step 1902 in the decision-based summarization process 1900 is to receive the input text. Although the embodiment described above uses sentences as the input text, any other text segment could be used instead, for example, clauses, paragraphs, or entire treatises.

Next, in step 1904, the input text is parsed to produce a syntactic tree in the style of Figure 11. If a full text is

used, one can use a discourse parse to build the discourse tree of the text.

In step 1906, the shift-reduce-drop algorithm is applied to the syntactic / discourse tree generated in step 1904. As discussed above, the shift-reduce-algorithm applies a sequence of predetermined decision rules (learned during training of the decision-based model, and identifying under what circumstances, and in what order, to perform the various shift-reduce-drop operations) to produce a compressed syntactic / discourse tree 1908. The resulting syntactic / discourse tree can be used for various purposes, for example, it can be rendered into a compressed text segment and output to a user (e.g., either a human end-user or a computer process). Alternatively, the resulting syntactic / discourse tree can be supplied to a process that further manipulates the tree for other purposes. For example, the resulting compressed syntactic / discourse tree could be supplied to a tree rewriter to convert it into another form, e.g., to translate it into a target language. An example of such a tree rewriter is described in Daniel Marcu et al., "The Automatic Translation of Discourse Structures," Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics, pp. 9-17, Seattle, Washington, (April 29 -- May 3, 2000).

Fig. 20 shows a generalized process for training a decision-based summarizer. As shown therein, the training process 2000 starts in step 2002 with an input training set as discussed above with reference to Fig. 17.

Next, in step 2004, the long-short text pairs are parsed to generate syntactic parse trees such as shown in Figure 11, thereby resulting in corresponding long-short syntactic tree pairs.

Next, in step 2006, for each long-short tree pair, the training process 2000 determines a sequence of shift-reduce-drop operations that will convert the long tree into the short tree. As discussed above, this step is performed based on the following four basic operations, referred to collectively as the "shift-reduce-drop" operations - shift, reduce, drop, and assignType. These four operations are sufficient to rewrite any given long tree into its paired short tree, provided that the order of the leaves does not change.

The output of step 2006 is a set of learning cases - one learning case for each long-short tree pair in the training set. In essence, each learning case is an ordered set of shift-reduce-drop operations that when applied to a long tree will generate the paired short tree.

Next, in step 2008, the training process 2000 associates features (e.g., operational and original-tree-specific features)

with the learning cases to reflect the context in which the operations are to be performed.

Next, in step 2010, the training process 2000 applies a learning algorithm, for example, the C4.5 algorithm as described in J. Ross Quinlan, "C4.5: Programs for Machine Learning," Morgan Kaufmann Publishers (1993), to learn a set of decision rules 2012 from the learning cases. This set of decision rules 2012 then can be used by the decision-based summarizer to summarize any previously unseen text or syntactic tree into a compressed version that is both coherent and grammatical.

Evaluation of the Summarizer Models

To evaluate the compression algorithms, 32 sentence pairs were randomly selected from the parallel corpus. This random subset is referred to as the *Test Corpus*. The other 1035 sentence pairs were used for training as described above. Figure 15 shows three sentences from the Test Corpus, together with the compressions produced by humans, the two compression algorithms described here (channel-based and decision-based), and a baseline algorithm that produces compressions with highest word-bigram scores. The examples were chosen so as to reflect good, average, and bad performance cases. The first sentence in Fig. 15 ("Beyond the basic level, the operations of the three products vary widely.") was compressed in the same manner by humans and by

both of the channel-based and decision-based algorithms (the baseline algorithm chooses though not to compress this sentence).

For the second example in Fig. 15, the output of the Decision-based algorithm is grammatical, but the semantics are negatively affected. The noisy-channel algorithm deletes only the word "break", which affects the correctness of the output less. In the last example in Fig. 15, the noisy-channel model is again more conservative and decides not to drop any constituents. In contrast, the decision-based algorithm compresses the input substantially, but it fails to produce a grammatical output.

Each original sentence in the *Test Corpus* was presented to four judges, together with four compressions of it: the human generated compression, the outputs of the noisy-channel and decision-based algorithms, and the output of the baseline algorithm. The judges were told that all outputs were generated automatically. The order of the outputs was scrambled randomly across test cases.

To avoid confounding, the judges participated in two experiments. In the first experiment, they were asked to determine on a scale from 1 to 5 how well the systems did with respect to selecting the most important words in the original sentence. In the second experiment, they were asked to determine on a scale from 1 to 5 how grammatical the outputs were.

It was also investigated how sensitive the channel-based and decision-based algorithms are with respect to the training data by carrying out the same experiments on sentences of a different genre, the scientific one. To this end, the first sentence of the first 26 articles made available in 1999 on the *cmplg* archive was used. A second parallel corpus, referred to as the *Cmplg* Corpus, was created by generating compressed grammatical versions of these sentences. Because some of the sentences in this corpus were extremely long, the baseline algorithm could not produce compressed versions in reasonable time.

The results of Table 5 show compression rate, and mean and standard deviation results across all judges, for each algorithm and corpus. The results show that the decision-based algorithm is the most aggressive: on average, it compresses sentences to about half of their original size. The compressed sentences produced by both the channel-based algorithm and by the decision-based algorithm are more "grammatical" and contain more important words than the sentences produced by the baseline. *T*-test experiments showed these differences to be statistically significant at $p < 0.01$ both for individual judges and for average scores across all judges. *T*-tests showed no significant statistical differences between the two algorithms. As Table 1 shows, the performance of the each of the compression algorithms is much closer to human performance than baseline performance;

yet, humans perform statistically better than our algorithms at $p < 0.01$.

Corpus	Avg. orig. sent. Length		Baseline	Noisy-channel	Decision-based	Humans
Test	21 words	Compression	63.70%	70.37%	57.19%	53.33%
		Grammaticality	1.78±1.19	4.34±1.02	4.30±1.33	4.92±0.18
		Importance	2.17±0.89	3.38±0.67	3.54±1.00	4.24±0.52
Cmplg	26 words	Compression	-	65.68%	54.25%	65.68%
		Grammaticality	-	4.22±0.99	3.72±1.53	4.97±0.08
		Importance	-	3.42±0.97	3.24±0.68	4.32±0.54

Table 5: Experimental results

When applied to sentences of a different genre, the performance of the noisy-channel compression algorithm degrades smoothly, while the performance of the decision-based algorithm drops sharply. This is due to a few sentences in the *Cmplg Corpus* that the decision-based algorithm over-compressed to only two or three words. This characteristic of the decision-based summarizer can be adjusted if the decision-based compression module is extended as described in D. Magerman, "Statistical decision-tree models for parsing," *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, 276-283 (1995), by computing probabilities across the sequences of decisions that correspond to a compressed sentence.

Similarly, noisy-channel modeling could be enhanced by taking into account subcategory and head-modifier statistics (in addition to simple word-bigrams). For example, the subject of a sentence may be separated from the verb by intervening

prepositional phrases. In this case, statistics should be collected over subject / verb pairs, which can be extracted from parsed text.

Although only a few embodiments have been described in detail above, those having ordinary skill in the art will certainly understand that many modifications are possible in the preferred embodiment without departing from the teachings thereof. All such modifications are encompassed within the following claims.